IN THE UNITED STATES PATENT & TRADEMARK OFFICE

In re the application of: Richards	Serial Number: 10/607,687
Filed: October 4, 2005	Attorney Docket No.: P103-US
Group Art Unit: 2677	Examiner: Shapiro, Leonid
Filed: October 4, 2005	

Title: **PREVENTION OF CHARGE ACCUMULATION IN MICROMIRROR DEVICES THROUGH BIAS INVERSION**

DECLARATION UNDER 37 C.F.R. §1.131

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

As below named inventors of the subject matter for which a United States Letters Patent is currently sought on the invention as identified above, I, Peter Richards, declare that:

1. I am the inventor of the claimed subject matter of the above-identified patent application; and

2. Prior to June 8, 2001 (the earliest effective filing date of the Markis reference US 6,724,379), I fully conceived the idea of a method of operating a micromirror device that comprises a movable mirror plate and an electrode formed on a substrate for driving the mirror plate, the method comprising: applying a first voltage to the mirror plate and a second voltage to the electrode such that voltage difference between the mirror plate and the electrode drives the mirror plate to rotate relative to the substrate; and applying a third voltage to the mirror plate, and a fourth voltage to the electrode such that the voltage difference between the mirror plate and the electrode drives the mirror plate to rotate

relative to the substrate, wherein difference between the third voltage and the fourth voltage has an opposite polarity to that between the first voltage and the second voltage.

For reducing the above idea into actual practice, a specific printed circuit board (PCB-000102 v1.00) was designed by me, Peter Richards; and fabricated by Hunter Technology. As shown in the attached Exhibit A of a copy of the purchase order, the printed circuit board described as: "PCB fabrication, PCB-000102 v1.00" was ordered on February 23, 2000. The purchase order was entered by the vendor on March 6, 2000.

Exhibit B shows a schematic diagram of the PCB-000102 v1.00 circuit board which performs functions including the invention as described in claim 1 in the above identified patent application. As highlighted in the first page of Exhibit B, an ordinary person skilled in the art will appreciate that signals LV_VBIAS, LV_VBORDER, BIAS_H-, BIAS_L, BIAS_OFFH, and BIAS_OFFL are designed to control the bias voltage as set forth in claim 1 of the above identified patent application. An exploded view of the schematic diagram in page one of Exhibition B is illustrated in page 2.

As can be seen in page 1 of Exhibition B, Xilinx XCV50-BG256 chip has designated IO signals of BIAS_H-, BIAS_L, BIAS_OFFH, and BIAS_L for controlling the bias voltage as set forth in claim 1 of the above identified patent application. The logic diagram showing the bias voltage drivers implemented in the system as shown in page 1 and page 2 is schematically illustrated in page 3 of Exhibit B. The portions of the Exhibit B highlighted in yellow make clear to one ordinary skilled in the art that the applicant was in possession of the invention.

In an exemplary implementation, program codes associated with the system in page 1, page 2, and page 3 are attached herewith as Exhibit C. The program codes were dated (accomplished) by May 11, 2000. These program codes implemented functions for controlling the bias voltages through parameters of bias_h, bias_l, bias_offl, and bias_offh, as shown on page 4; and the logic expression of "wire [63:0] rbuf_dinv=rbuf_inv ? ~rbif_din : rbuf_din" on page 7 of Exhibit C.

Applicant believes that Exhibits A, B, and C each describe a definite and permanent idea of the complete and operative invention as set forth in claim 1 of the above identified patent application. It is also believed that, if presented to one ordinary skilled in the art, the information set forth in the attached exhibits, combined with background knowledge in the art would allow one of ordinary skill in the art to proceed to actually reduce the conceived invention into practice make and use the invention set forth in the claims pending in the above-identified patent application, without having to supply an unobvious contribution.

Though the pages containing the diagrams in Exhibit B and program codes in Exhibit C are unsigned (Reflectivity was a very small company at that time and did not always follow "best practices" for laboratory notebooks or records), each page in Exhibit B has an electronic date – the date when the diagram in that page was last modified. The header of the program codes as shown in Exhibit C has an electronic signature containing the date when the program codes were finished.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or by both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of sole or first inventor: **PETER, RICHARDS**

Inventor's signature: [illegible]
Date: 1/25/2006
Citizenship: USA
Residence and P.O. Address:
994 CAROLINA ST
SAN FRANCISCO CA 94107

Exhibit A

Reflectivity, Inc.

3910 Freedom Circle, Suite 103
Santa Clara, CA 95054
(408) 970 - 8881 fax (408) 970 - 8840

Purchase Order No. PR004012

PURCHASE ORDER

Vendor

Name	Hunter Technology
	3305 Kifer Rd
	Santa Clara, CA 95051
Phone	800 570 8946
Fax	408 736 1908

Ship To

Name	Reflectivity, Inc.
Address	3910 Freedom Circle, Suite 103
City	Santa Clara St CA ZIP 95054
Phone	(408) 970 - 8881

Qty	Units	Description	Unit Price	TOTAL
10	ea	PCB fabrication, PCB-000102 v1.00 5 day turn	$185.00	$1,850.00
1		NRE	$160.00	$160.00
1		Test	$100.00	$100.00
1		Fixture	$300.00	$300.00
50	ea	PCB fabrication, PCB-000102 v1.00 10 day turn Note: do not proceed w/ fab of second lot until authorization from Reflectivity Hunter Tech contact: David Manley 408 328 9707	$34.00	$1,700.00

SubTotal	$4,110.00
Shipping & Handling	
Taxes State	
TOTAL	$4,110.00

Payment Details

- Check
- Cash
- [✓ Account No.]
- Credit Card

Name

CC #

Exp Date

Shipping Date

Approval

Date	2/23/00
Order No	
Sales Rep	
Ship Via	

Notes/Remarks

Ref. Quote dated 2/16
Ordered By: PR Account: 7754 Demo System

HUNTER TECHNOLOGY
3305 Kifer Road
Santa Clara, CA 95051
TEL 800.570.8946 FAX 408.736.1908

Invoice No 0000012335

Customer 001435

Bill to :

REFLECTIVITY, INC
ATTN ACCOUNTS PAYABLE
3910 FREEDOM CIRCLE
SUITE # 103
SANTA CLARA CA 95054

Sold to :

REFLECTIVITY, INC
ATTN ACCOUNTS PAYABLE
3910 FREEDOM CIRCLE
SUITE # 103
SANTA CLARA CA 95054

Phone (408) 970-8881

Customer PO Number	Invoice Date	Terms	FOB	Ship Via	Salesperson
PR004012	02/23/2000	C.O.D.	OUR PLANT	WILL CALL	DPM

Item	Part / Rev / Description / Details	Quantity	Unit Price	Discount	Extended Price
000001	PCB-000102 Rev NS U/M EA PCB FABRICATION	10.00	185.00000	0.00	1,850.00
000011	LOCAL SALES TAX Rev Sales Tax to SANTA CLARA SALES TAX for Line Item No 000001. Calculated at 8.250 percent of the extended price as of 02/24/2000.	1.00	152.63000	0.00	152.63
000003	NRE Rev 00 U/M EA ONE TIME CHARGE - FAB	1.00	160.00000	0.00	160.00
000021	LOCAL SALES TAX Rev Sales Tax to SANTA CLARA SALES TAX for Line Item No 000003. Calculated at 8.250 percent of the extended price as of 02/24/2000.	1.00	13.20000	0.00	13.20
000004	TEST Rev U/M EA TEST - FAB	1.00	100.00000	0.00	100.00
000041	LOCAL SALES TAX Rev Sales Tax to SANTA CLARA SALES TAX for Line Item No 000004. Calculated at 8.250 percent of the extended price as of 02/24/2000.	1.00	8.25000	0.00	8.25
000005	FIXTURE Rev U/M EA FIXTURE - FAB	1.00	300.00000	0.00	300.00
000051	LOCAL SALES TAX Rev Sales Tax to SANTA CLARA SALES TAX for Line Item No 000005. Calculated at 8.250 percent of the extended price as of 02/24/2000.	1.00	24.75000	0.00	24.75

ENTERED
MAR 06 2000

CUSTOMER COPY

HUNTER TECHNOLOGY
3305 Kifer Road
Santa Clara, CA 95051
TEL 800.570.8946 FAX 408.736.1908

Shipping List 010878
Customer No 001435
Sales Order Shipper

Ship to : REFLECTIVITY, INC
3910 FREEDOM CIRCLE
SUITE # 103
SANTA CLARA, CA 95054

Sold to : REFLECTIVITY, INC
ATTN ACCOUNTS PAYABLE
3910 FREEDOM CIRCLE
SUITE # 103
SANTA CLARA, CA 95054

COD Shipment!

Ship Date	Customer PO	Sales Order	# of Boxes	Weight	Ship VIA	Bill of Lading	F O B
02/23/2000	PR004012	007660-00	1	.0000	WILL CALL		OUR PLANT

Item	Part / Rev / Description / Details	Order Quantity	Ship Quantity
000001	PCB-000102 U/M EA SO Item 1 PCB FABRICATION	10.00	10.00

[illegible]

CUSTOMER COPY

HUNTER TECHNOLOGY
3305 Kifer Road
Santa Clara, CA 95051
TEL 800.570.8946 FAX 408.736.1908

Invoice No 0000012335

Customer 001435

Bill to :

REFLECTIVITY, INC
ATTN ACCOUNTS PAYABLE
3910 FREEDOM CIRCLE
SUITE # 103
SANTA CLARA CA 95054

Sold to :

REFLECTIVITY, INC
ATTN ACCOUNTS PAYABLE
3910 FREEDOM CIRCLE
SUITE # 103
SANTA CLARA CA 95054

Phone (408) 970-8881

Customer PO Number	Invoice Date	Terms	FOB	Ship Via	Salesperson
PR004012	02/23/2000	C.O.D.	OUR PLANT	WILL CALL	DPM

Item	Part / Rev / Description / Details	Quantity	Unit Price	Discount	Extended Price
000001	PCB-000102 Rev NS U/M EA PCB FABRICATION	10.00	185.00000	0.00	1,850.00
000011	LOCAL SALES TAX Rev Sales Tax to SANTA CLARA SALES TAX for Line Item No 000001. Calculated at 8.250 percent of the extended price as of 02/24/2000.	1.00	152.63000	0.00	152.63
000003	NRE Rev 00 U/M EA ONE TIME CHARGE - FAB	1.00	160.00000	0.00	160.00
000021	LOCAL SALES TAX Rev Sales Tax to SANTA CLARA SALES TAX for Line Item [illegible] Calculated at 8.250 percent of the extended	1.00	13.20000	0.00	13.20

REFLECTIVITY, INC

Hunter Technology
03/01/2000

2248

Bill #12335

3/16/2000

2,608.83

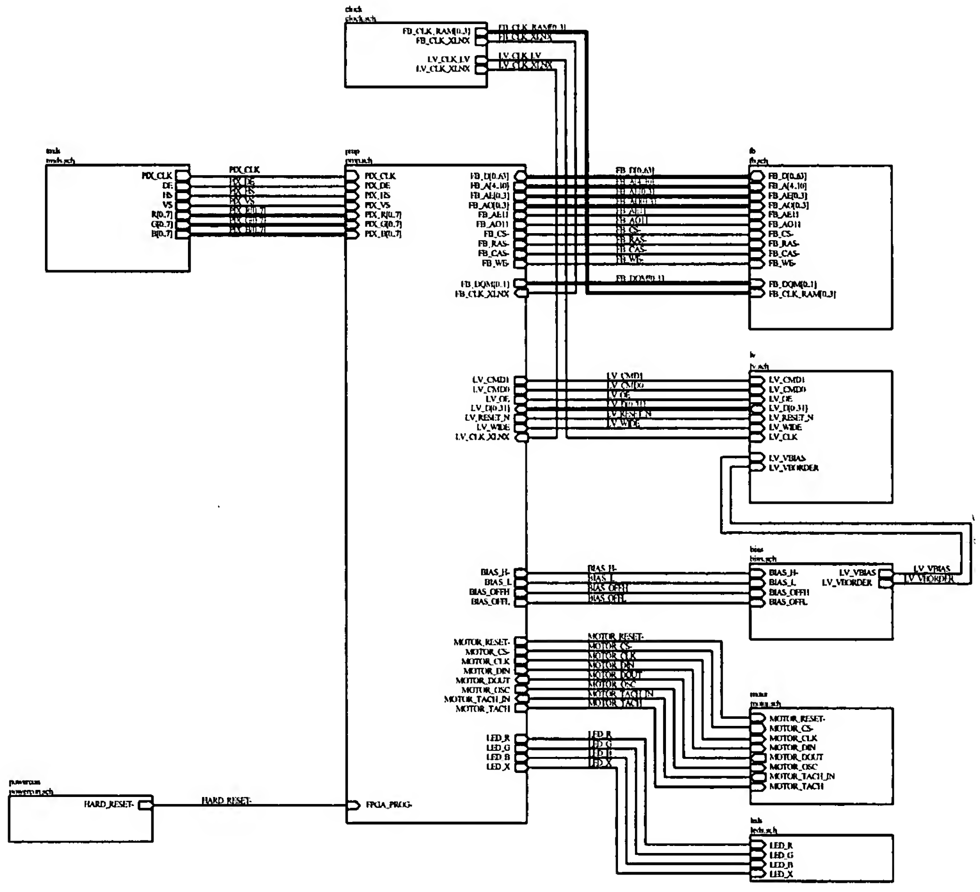
PAYMENT RECORD

Cash - Silicon Valley Chec Inv# 12335, PO# PR004012

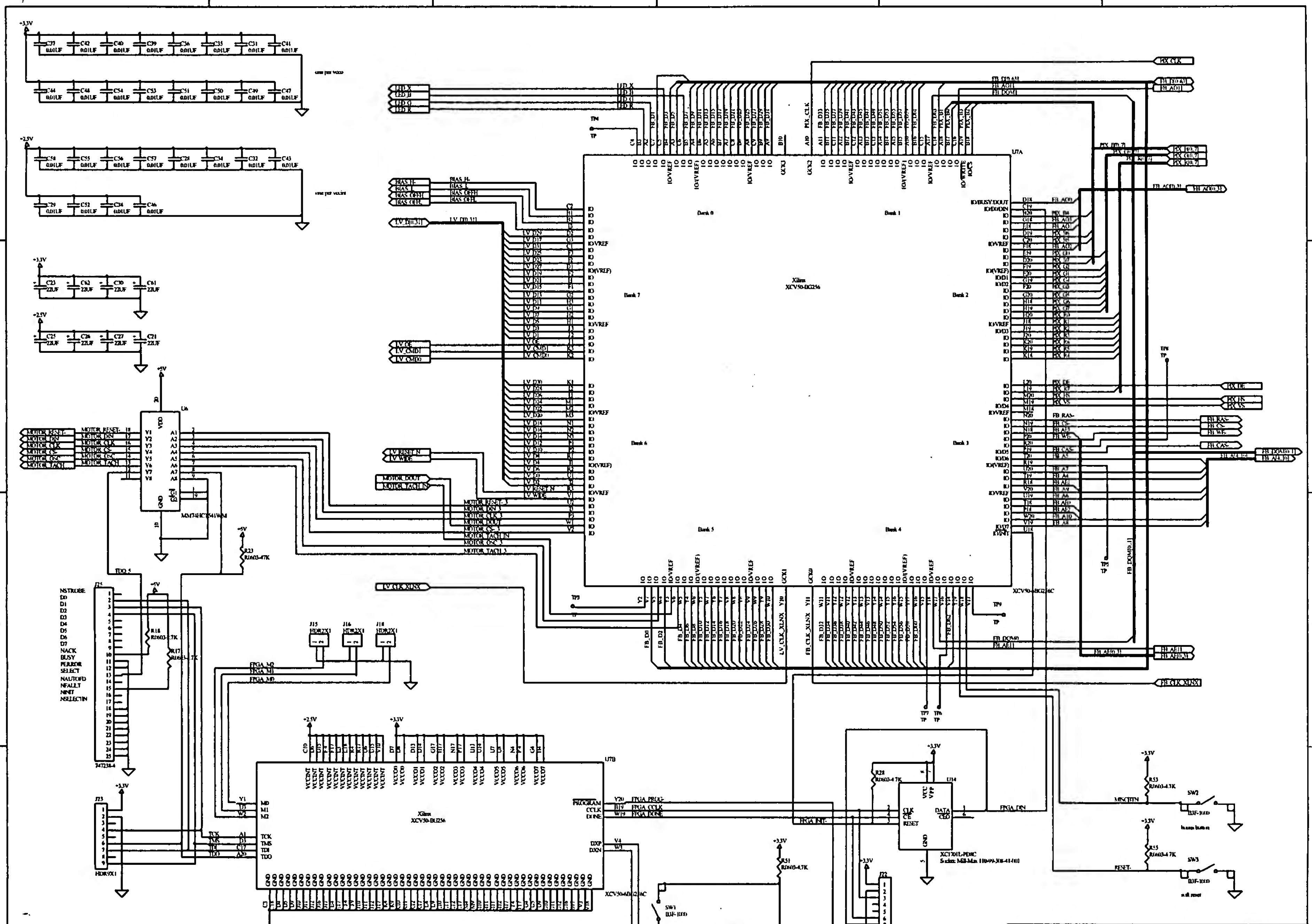
2,608.83

126898 (8/99)

Exhibit B

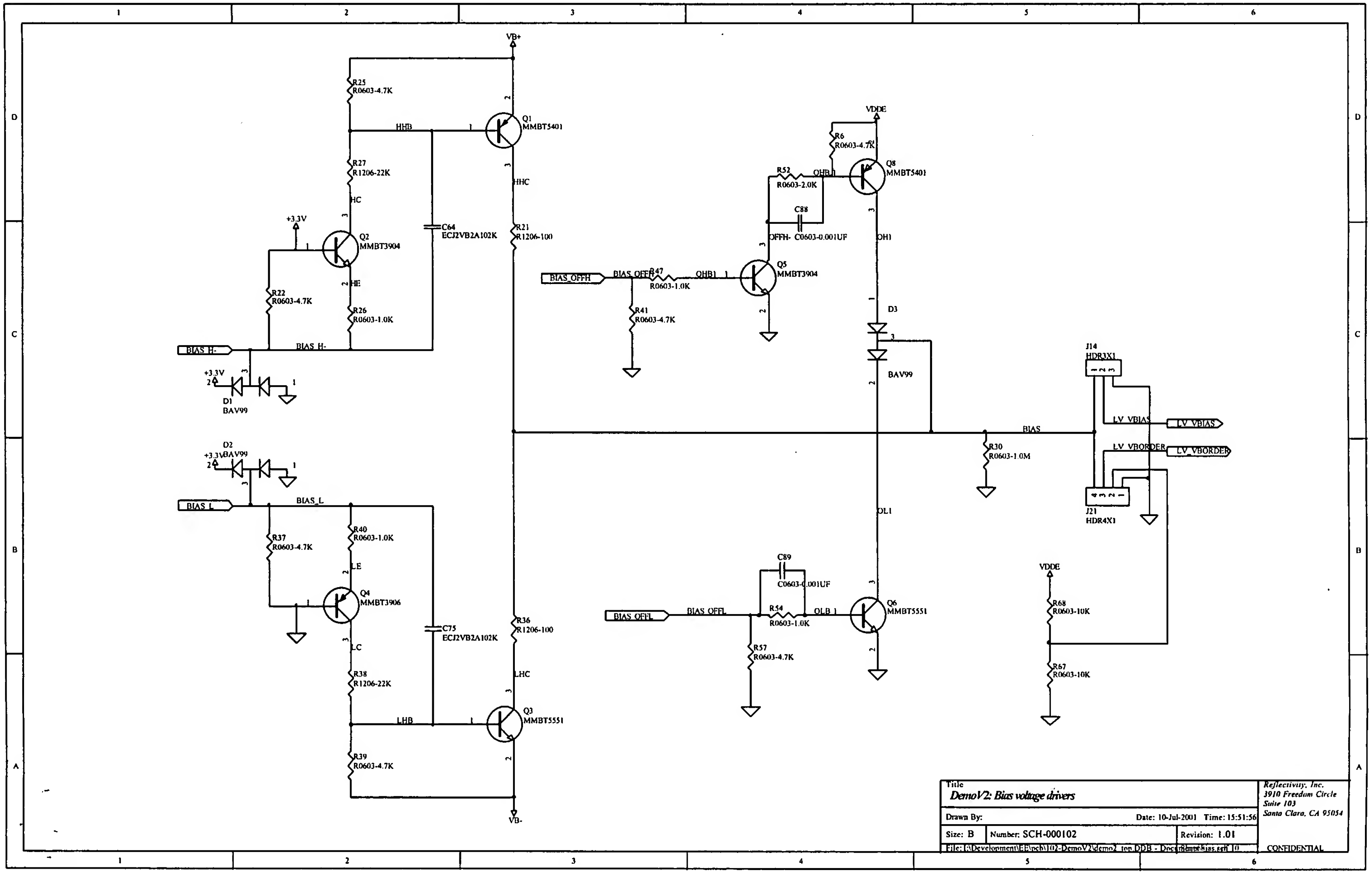


Title			Replicatory, Inc. 3910 Freedom Circle Suite 103 Santa Clara, CA 95054
DemoV2: Top-level schematic			
Drawn By:	Date: 10-Jul-2001	Time: 15:51:42	
Size: C	Number: SCH-000102	Revision: 1.01	
File: [illegible]		Sheet [illegible] of 10	CONFIDENTIAL



Title
DemoV2: Pixel modulation processor FPGA

Drawn By: P. Richards	Date: 10-Jul-2001	Time: 15:51:44	Reflectivity, Inc. 3910 Freedom Circle Suite 103 Santa Clara, CA 95054
Size: C	Number: SCH-000102	Revision: 1.01	
File: [illegible]	Sheet 2 of 10		CONFIDENTIAL



Title
DemoV2: Bias voltage drivers

Drawn By:

Date: 10-Jul-2001 Time: 15:51:56

Size: B Number: SCH-000102 Revision: 1.01

File: I:\Development\EE\pcb\102-DemoV2\demo2_top.DDB - Documents\bias.sch 10

Reflectivity, Inc.
3910 Freedom Circle
Suite 103
Santa Clara, CA 95054

CONFIDENTIAL

Exhibit C

```
/*
lv_engine.v

2000 Reflectivity, Inc. CONFIDENTIAL

$Id: lv_engine.v,v 1.10 2000/05/11 01:12:37 cvs Exp $

Manages interface to LV, timing of events on LV bus, and keeps
LEDs and bias in sync with the operations happening in the LV.
*/

`include "b2.vh"

module lv_engine(
  reset, clk,

  queue_empty, queue_pull, lvq_x, lv_queue_dout, lv_flags,

  lv_reset_n, lv_wide, lv_cmd, lv_oe, lv_doe_n, lv_dout, lv_din,

  led_r, led_g, led_b, led_x,

  bias_h, bias_l, bias_offh, bias_offl
);
input reset;
input clk;

input queue_empty;
output queue_pull;
output [5:0] lvq_x;
input [31:0] lv_queue_dout;
input [7:0] lv_flags;

output lv_reset_n;
output lv_wide;
output [1:0] lv_cmd;
output lv_oe;
output [31:0] lv_doe_n;
output [31:0] lv_dout;
input [31:0] lv_din;

output led_r;
output led_g;
output led_b;
output led_x;
output bias_h;
output bias_l;
output bias_offh;
output bias_offl;

////
// Update rate timer
//
// The LV controller's responsibility is to issue row writes
// to the actual lightvalve based on the data and commands that
// appear in the LV queue.
// Each data+command in the queue specifies a delay count; the LV
// controller times the operations on the LV bus operations such
// that the current row update and the next row update are spaced
// apart by this delay value.
//
wire [15:0] lv_rate = 16'd52; // XXX hardcoded for now
reg [15:0] lv_timer;
reg lv_timer_last;
always @(posedge clk or posedge reset) begin
  if (reset) begin
    lv_timer <= 0;
    lv_timer_last <= 1;
    end
    else begin
      // to reduce combinational delay for logic that depends on
      // lv_timer state, generate registered
      // version of flag indicating that lv_timer == 0
      if ( (lv_timer == 1)
           || (lv_timer == 0) && queue_empty )
        // lv_timer is going to be 0 on next cycle
        lv_timer_last <= 1;
      else
        lv_timer_last <= 0;

      if ( lv_timer != 0 ) begin
        // timer is active, let it count down
        lv_timer <= lv_timer - 1;
      end
      else begin // lv_timer == 0
        // timer has expired, trigger next update and
        // restart the timer, *if* there's something
        // waiting in the queue.  Otherwise wait until
        // a row is ready in the queue so we have a valid
        // reload value for the timer.
        if ( ! queue_empty ) begin
          // reload
          lv_timer <= lv_rate; // XXX select based on flags
        end
      end
    end
  end

  ////
  // Queue fetch address
  //
  // Whenever the timer has expired, a new update operation
  // begins (assuming the queue is non-empty!).  Fetch
  // the appropriate LV bus word from the queue.
  // Currently the queue is arranged such that
  // addresses 00-1f contain the 32 words of pixel data, and the
  // command word containing the write_row command is stored
  // in address 20.
  //
  reg [5:0] lvq_x;
  wire lvq_x_last = lvq_x == 'h20;
  wire lv_trigger = lv_timer_last && !queue_empty;
  wire queue_pull = lvq_x_last;
  always @(posedge clk or posedge reset) begin
    if (reset) begin
      lvq_x <= 'h30; // start off in kludgy initialization state
    end
    else begin
      if ( (lvq_x != 0) || lv_trigger ) begin
        lvq_x <= lvq_x_last ? 0 : (lvq_x + 1);
      end
    end
  end

  ////
  // LV control signals
  //
  // pipelined to match delay from lvq_x -> lvq output -> i/o flop
  reg [1:0] cmd;
  reg oe; // control signal enabling LV to drive bus
  reg doe; // control signal enabling *our* bus drivers
  reg [7:0] save_flags;
  always @(posedge clk or posedge reset) begin
    if (reset) begin
      cmd <= `LVCMD_IDLE;
      oe <= 0;
      doe <= 0;
      save_flags <= 0;
    end
    else begin
      if ( lvq_x == 0 ) begin
        if ( !lv_trigger ) begin
          // lv_trigger is false; no data is available, lvq_x is
          // going to stay at 0 next cycle, and we should idle
          cmd <= `LVCMD_IDLE;
        end
        else begin // lv_trigger
          // update operation is beginning.  lvq_x is currently 0
          // and the corresponding data will appear on lv_queue_dout
          // next cycle; set cmd to LVCMD_DATA to match
          cmd <= `LVCMD_DATA;
        end
      end // lvq_x == 0
      else if ( lvq_x <= 'h1f ) begin
        // data cycle
        cmd <= `LVCMD_DATA;
      end
      else if ( lvq_x == 'h20 ) begin
        // write_row command at end of cycle
        cmd <= `LVCMD_CTRL;
        save_flags <= lv_flags;
      end
      // hack to write config reg on initialization
      else if ( lvq_x == 'h3e ) begin
        cmd <= `LVCMD_CTRL;
      end
      else if ( lvq_x == 'h3f ) begin
        cmd <= `LVCMD_CDATA;
      end
      else begin
        cmd <= `LVCMD_IDLE;
      end

      oe <= 0;   // LV is write-only unless we're testing it...
      doe <= 1;  // ...we always own the bus
    end
  end

  ////
  // LV IOB logic
  //
  wire lv_reset_n = !reset; // XXX need to do real initialization
  wire lv_wide = 0;
  reg [1:0] lv_cmd;
  reg lv_oe;
  wire [31:0] lv_din;
  reg [31:0] lv_dout;
  reg [31:0] lv_doe_n;
  always @(posedge clk or posedge reset) begin
    if (reset) begin
      lv_cmd <= `LVCMD_IDLE;
      lv_oe <= 0;
      lv_dout <= 0;
      lv_doe_n <= ~32'b0;
    end
    else begin
      lv_cmd <= cmd;
      lv_oe <= oe;
      lv_dout <= lv_queue_dout;
      lv_doe_n <= doe ? 32'b0 : ~32'b0;
    end
  end

  ////
```

```
// Lightvalve initialization parameters
//
wire b2_config_dr = 0;
wire b2_config_dl = 0;
wire b2_config_db = 0;
wire b2_config_dt = 0;
wire b2_config_rbt = 1;
wire [2:0] b2_config_wbc = 3'b011;
wire [2:0] b2_config_wwc = 3'b011;
wire [3:0] b2_config_rbc = 4'b0111;
wire b2_config_rsc = 1;
wire b2_config = {
  16'b0,
  b2_config_rsc, b2_config_rbc,
  b2_config_wwc, b2_config_wbc,
  b2_config_rbt,
  b2_config_dt, b2_config_db,
  b2_config_dl, b2_config_dr
};

////
// dummy led and bias control for now
// synchronize led/bias update to actual timing of write event
// in lightvalve
// delay is 4 cycles + wait states from write_row command.  Add 2 for
// latency from flag_trig to actual pins, minus 1 because SRL delay element
// gives you n+1 cycles of delay...adds up to the formula below for
// delay_count

wire flag_trig = lvq_x == 'h20;
wire flag_trig_delayed;
wire [3:0] delay_count = 4'd4 + b2_config_wbc + b2_config_wwc + 1;
prim_srl16e flag_trig_delay(
  .clk( clk ), .ce( 1'b1 ), .a( delay_count ),
  .d( flag_trig ), .q( flag_trig_delayed )
);

reg led_r;
reg led_g;
reg led_b;
reg led_x;

reg bias_h;
reg bias_l;
reg bias_offh;
reg bias_offl;

always @(posedge clk or posedge reset) begin
  if (reset) begin
    led_r <= 0; led_g <= 0; led_b <= 0; led_x <= 0;
    bias_h <= 0; bias_l <= 0; bias_offh <= 0; bias_offl <= 0;
  end
  else begin
    if (flag_trig_delayed) begin
      case (save_flags[1:0]) // LEDs
        0: begin
          led_r <= 1; led_g <= 0; led_b <= 0; led_x <= 0;
        end

        1: begin
          led_r <= 0; led_g <= 1; led_b <= 0; led_x <= 0;
        end

        2: begin
          led_r <= 0; led_g <= 0; led_b <= 1; led_x <= 0;
        end

        3: begin
        led_r <= 0; led_g <= 0; led_b <= 0; led_x <= 0;
      end
    endcase

    case (save_flags[3:2]) // bias
      0: begin
        bias_h <= 1; bias_l <= 0; bias_offh <= 0; bias_offl <= 0;
      end

      1: begin
        bias_h <= 0; bias_l <= 0; bias_offh <= 0; bias_offl <= 1;
      end

      2: begin
        bias_h <= 0; bias_l <= 1; bias_offh <= 0; bias_offl <= 0;
      end

      3: begin
        bias_h <= 0; bias_l <= 0; bias_offh <= 1; bias_offl <= 0;
      end
    endcase
  end
 end
end

endmodule
```

```
/*
lv_queue.v

2000 Reflectivity, Inc. CONFIDENTIAL

$Id: lv_queue.v,v 1.5 2000/05/11 01:12:37 cvs Exp $

Manages queue of data and commands to lightvalve
*/

`include "b2.vh"

module lv_queue(
  reset, lv_clk, fb_clk,

  pwm_ready, pwm_ack,
  pwm_b, pwm_y, pwm_pe, pwm_po, pwm_flags,

  read_req_set, read_ready,
  read_b, read_y, read_pe, read_po, read_tag,

  rbuf_din, rbuf_valid, rbuf_complete, rbuf_tag, rbuf_x,

  lv_queue_empty, lv_queue_pull, lvq_x, lv_queue_dout, lv_flags
);
input reset;
input lv_clk;
input fb_clk;

input pwm_ready;
output pwm_ack;
input pwm_b;
input [11:0] pwm_y;
input [5:0] pwm_pe;
input [5:0] pwm_po;
input [7:0] pwm_flags;

output read_req_set;
input read_ready;
output read_b;
output [11:0] read_y;
output [5:0] read_pe;
output [5:0] read_po;
output [3:0] read_tag;

input [63:0] rbuf_din;
input rbuf_valid;
input rbuf_complete;
input [3:0] rbuf_tag;
input [5:0] rbuf_x;

output lv_queue_empty;
input lv_queue_pull;
input [5:0] lvq_x;
output [31:0] lv_queue_dout;
output [7:0] lv_flags;

////
// post requests to sdram interface as prompted by the pwm controller
//
reg read_req_set;
reg read_ready1;
wire read_ack = read_ready && !read_ready1;
wire pwm_ack = read_ack;
wire read_b = pwm_b;
wire [11:0] read_y = pwm_y;
wire [5:0] read_pe = pwm_pe;
wire [5:0] read_po = pwm_po;
wire [3:0] read_tag;
wire lv_queue_full;
always @(posedge lv_clk or posedge reset) begin
  if (reset) begin
    read_req_set <= 0;
    read_ready1 <= 1;
  end
  else begin
    read_ready1 <= read_ready;
    read_req_set <= read_ready && pwm_ready && !lv_queue_full && !read_req_set;
  end
end


////
// keep track of queue state
//
// queue_head:
//   location from which next queue element will be pulled
//   this location may or may not contain valid data...it's
//   not ready unless queue_fill != queue_head
//
// queue_tail:
//   location of next empty queue slot.  In the queue-full
//   condition points to the head of the queue, where the next
//   item will go as soon as that slot becomes available
//
// queue_fill:
//   location of queue slot currently being filled.  Different
//   from queue_tail in that queue_tail is bumped when a slot *begins*
//   to get filled, and queue_tail is bumped when a slot is *finished* being
//   filled. (ok, *almost* finished...bumped when frame buffer read is
//   acknowledged, at which point there are still a few clock cycles
//   to go...resort to extra flag to keep track of this boundary case)
//
reg [2:0] queue_head;
reg [2:0] queue_tail;
reg [2:0] queue_fill;
assign read_tag = { 1'b0, queue_fill };
wire [2:0] queue_almost_full_sub = queue_head - queue_tail;
wire queue_almost_full = queue_almost_full_sub == 1;
wire [2:0] queue_almost_empty_sub = queue_fill - queue_head;
wire queue_almost_empty = queue_almost_empty_sub == 1;
reg queue_full;
reg queue_empty_flag;
wire queue_empty;
wire queue_pull = lv_queue_pull;
wire lv_queue_empty = queue_empty;
assign lv_queue_full = queue_full;
always @(posedge lv_clk or posedge reset) begin
  if (reset) begin
    queue_head <= 0;
    queue_tail <= 0;
    queue_fill <= 0;
    queue_full <= 0;
    queue_empty_flag <= 1;
  end
  else begin
    if (read_req_set) begin
      queue_tail <= queue_tail + 1;
    end

    if (read_ack) begin
      queue_fill <= queue_tail;
    end

    if (queue_pull) begin
      queue_head <= queue_head + 1;
    end

    if ( queue_almost_full && read_req_set && !queue_pull ) begin
      queue_full <= 1;
    end
    else if ( !read_req_set && queue_pull ) begin
      queue_full <= 0;
    end

    if ( queue_almost_empty && queue_pull && !read_ack ) begin
      queue_empty_flag <= 1;
    end
    else if ( !queue_pull && read_ack ) begin
      queue_empty_flag <= 0;
    end

  end
end

// kludgy flag to keep track of the fact that a read request has
// been acknowledged (and a new one can be issued) but the actual data
// hasn't all been delivered yet...
//
// queue_last_busy indicates that, if (queue_fill - queue_head) == 1,
// the one item in the queue isn't quite ready yet, and the queue should
// temporarily be considered 'empty' to avoid reading the data prematurely

reg queue_last_busy;
assign queue_empty = queue_empty_flag || (queue_almost_empty && queue_last_busy);
always @(posedge lv_clk or posedge reset or posedge rbuf_complete) begin
  if (reset)
    queue_last_busy <= 0;
  else if (rbuf_complete)
    queue_last_busy <= 0; // async reset
  else if (read_ack)
    queue_last_busy <= 1;
end

////
// Remember the row index associated with request in progress
reg [11:0] rbuf_y;
always @(posedge lv_clk or posedge reset) begin
  if (reset) begin
    rbuf_y <= 0;
  end
  else begin
    if ( read_ack ) rbuf_y <= read_y;
  end
end

////
// Stuff queue with (usually) pixel data or (sometimes) a
// command word
wire rbuf_inv = 0;
wire [63:0] rbuf_dinv = rbuf_inv ? ~rbuf_din : rbuf_din;
wire [63:0] rbuf_d =
  rbuf_complete ? { rbuf_dinv[63:32], `LVREG_WRITE_ROW, 16'b0, rbuf_y }
                : rbuf_dinv ;

////
// Buffer
// accept frame buffer data (up to 8 rows) from the sdram controller
// and store in preparation to send out the lv bus

wire [7:0] lvq_w_addr = { rbuf_tag[2:0], rbuf_x[4:0] };
wire lvq_we = rbuf_valid || rbuf_complete;
wire [8:0] lvq_r_addr = { queue_head, lvq_x };
```

```
lv_data_buf lv_data_buf (
  .reset( reset ), .wclk( fb_clk ),
  .we( lvq_we ), .waddr( lvq_w_addr ), .d( rbuf_d ),
  .rclk( lv_clk ), .raddr( lvq_r_addr ), .q( lv_queue_dout )
);

////
// Flag buffer
// remember auxiliary info associated with each row
lv_flag_buf lv_flag_buf (
  .reset( reset ), .wclk( lv_clk ),
  .we( read_ack ), .waddr( {1'b0, queue_fill} ), .d( pwm_flags ),
  .rclk( lv_clk ), .raddr( {1'b0, queue_head} ), .q( lv_flags )
);

endmodule

////
// lv_data_buf
//
// actual RAM buffer portion of the LV queue, stores up to
// 8 rows of image bits.
// Also holds command words so we can avoid putting any muxes
// in the RAM->IOB path

module lv_data_buf (
  reset, wclk,
  we, waddr, d,
  rclk, raddr, q
);
input reset;
input wclk;
input we;
input [7:0] waddr;
input [63:0] d;
input rclk;
input [8:0] raddr;
output [31:0] q;

// split input data into even/odd halves
wire [31:0] de =
  { d[62], d[60], d[58], d[56],
    d[54], d[52], d[50], d[48],
    d[46], d[44], d[42], d[40],
    d[38], d[36], d[34], d[32],
    d[30], d[28], d[26], d[24],
    d[22], d[20], d[18], d[16],
    d[14], d[12], d[10], d[ 8],
    d[ 6], d[ 4], d[ 2], d[ 0] };

wire [31:0] do =
  { d[63], d[61], d[59], d[57],
    d[55], d[53], d[51], d[49],
    d[47], d[45], d[43], d[41],
    d[39], d[37], d[35], d[33],
    d[31], d[29], d[27], d[25],
    d[23], d[21], d[19], d[17],
    d[15], d[13], d[11], d[ 9],
    d[ 7], d[ 5], d[ 3], d[ 1] };

wire [15:0] qe;
wire [15:0] qo;

////
// initialization hack
// stuff words for the CONFIG register write into
// the queue on startup
//
```

```
reg [1:0] qinit_state;
reg [8:0] raddr_init;
reg [31:0] qinit;
reg qinit_we;
wire [15:0] qinit_e = {
  qinit[30], qinit[28], qinit[26], qinit[24],
  qinit[22], qinit[20], qinit[18], qinit[16],
  qinit[14], qinit[12], qinit[10], qinit[ 8],
  qinit[ 6], qinit[ 4], qinit[ 2], qinit[ 0] };
wire [15:0] qinit_o = {
  qinit[31], qinit[29], qinit[27], qinit[25],
  qinit[23], qinit[21], qinit[19], qinit[17],
  qinit[15], qinit[13], qinit[11], qinit[ 9],
  qinit[ 7], qinit[ 5], qinit[ 3], qinit[ 1] };

always @(qinit_state or raddr) begin
  case (qinit_state)
    0: begin
      raddr_init <= 9'h03e;
      qinit_we <= 0;
      qinit <= { `LVREG_CONFIG, 28'b0 };
    end
    1: begin
      raddr_init <= 9'h03e;
      qinit_we <= 1;
      qinit <= { `LVREG_CONFIG, 28'b0 };
    end
    2: begin
      raddr_init <= 9'h03f;
      qinit_we <= 1;
      qinit <= 32'h0000bb70;
    end
    3: begin
      raddr_init <= raddr;
      qinit_we <= 0;
      qinit <= 32'h0000bb70;
    end
  endcase
end

always @(posedge rclk or posedge reset) begin
  if (reset) begin
    qinit_state <= 2'b00;
  end
  else begin
    if ( qinit_state != 2'b11 ) qinit_state <= qinit_state + 1;
  end
end

RAMB4_S8_S16 ram_e0(
  .CLKB( wclk ), .WEB( we ), .ENB( 1'b1 ), .RSTB( 1'b0 ),
  .ADDRB( waddr ), .DIB( { de[23:16], de[7:0] } ), .DOB( ),
  .CLKA( rclk ), .WEA( qinit_we ), .ENA( 1'b1 ), .RSTA( 1'b0 ),
  .ADDRA( raddr_init ), .DIA( qinit_e[7:0] ), .DOA( qe[7:0] )
);

RAMB4_S8_S16 ram_e1(
  .CLKB( wclk ), .WEB( we ), .ENB( 1'b1 ), .RSTB( 1'b0 ),
  .ADDRB( waddr ), .DIB( { de[31:24], de[15:8] } ), .DOB( ),
  .CLKA( rclk ), .WEA( qinit_we ), .ENA( 1'b1 ), .RSTA( 1'b0 ),
  .ADDRA( raddr_init ), .DIA( qinit_e[15:8] ), .DOA( qe[15:8] )
);

RAMB4_S8_S16 ram_o0(
  .CLKB( wclk ), .WEB( we ), .ENB( 1'b1 ), .RSTB( 1'b0 ),
  .ADDRB( waddr ), .DIB( { do[23:16], do[7:0] } ), .DOB( ),
  .CLKA( rclk ), .WEA( qinit_we ), .ENA( 1'b1 ), .RSTA( 1'b0 ),
  .ADDRA( raddr_init ), .DIA( qinit_o[7:0] ), .DOA( qo[7:0] )
```

```
);

RAMB4_S8_S16 ram_o1(
  .CLKB( wclk ), .WEB( we ), .ENB( 1'b1 ), .RSTB( 1'b0 ),
  .ADDRB( waddr ), .DIB( { do[31:24], do[15:8] } ), .DOB( ),
  .CLKA( rclk ), .WEA( qinit_we ), .ENA( 1'b1 ), .RSTA( 1'b0 ),
  .ADDRA( raddr_init ), .DIA( qinit_o[15:8] ), .DOA( qo[15:8] )
);



wire [31:0] q = {
  qo[15], qe[15], qo[14], qe[14],
  qo[13], qe[13], qo[12], qe[12],
  qo[11], qe[11], qo[10], qe[10],
  qo[ 9], qe[ 9], qo[ 8], qe[ 8],
  qo[ 7], qe[ 7], qo[ 6], qe[ 6],
  qo[ 5], qe[ 5], qo[ 4], qe[ 4],
  qo[ 3], qe[ 3], qo[ 2], qe[ 2],
  qo[ 1], qe[ 1], qo[ 0], qe[ 0]
};

endmodule

////
// lv_flag_buf
//
// stores flags associated with each row in the queue.
// currently there are 8 bits of flags; these represent:
//   7: double-buffer bank
//   6-5: selects delay until next write
//   4: invert data
//   3-2: bias to take effect upon *this* write
//   1-0: led state to take effect upon *this* write
//
// also keeps track of flags indicating which rows in the
// lv queue are valid.
//
module lv_flag_buf (
  reset, wclk,
  we, waddr, d,
  rclk, raddr, q
);
input reset;
input wclk;
input we;
input [3:0] waddr;
input [7:0] d;
input rclk;
input [3:0] raddr;
output [7:0] q;

wire [7:0] q1;

// dual-port RAM to hold flags
prim_ram16x1d flag0(
  .wclk( wclk ), .we( we ), .a( waddr ),
  .d( d[0] ), .spo( ),
  .dpra( raddr ), .dpo( q1[0] )
);

prim_ram16x1d flag1(
  .wclk( wclk ), .we( we ), .a( waddr ),
  .d( d[1] ), .spo( ),
  .dpra( raddr ), .dpo( q1[1] )
);

prim_ram16x1d flag2(
  .wclk( wclk ), .we( we ), .a( waddr ),
  .d( d[2] ), .spo( ),
  .dpra( raddr ), .dpo( q1[2] )
);

prim_ram16x1d flag3(
  .wclk( wclk ), .we( we ), .a( waddr ),
  .d( d[3] ), .spo( ),
  .dpra( raddr ), .dpo( q1[3] )
);

prim_ram16x1d flag4(
  .wclk( wclk ), .we( we ), .a( waddr ),
  .d( d[4] ), .spo( ),
  .dpra( raddr ), .dpo( q1[4] )
);

prim_ram16x1d flag5(
  .wclk( wclk ), .we( we ), .a( waddr ),
  .d( d[5] ), .spo( ),
  .dpra( raddr ), .dpo( q1[5] )
);

prim_ram16x1d flag6(
  .wclk( wclk ), .we( we ), .a( waddr ),
  .d( d[6] ), .spo( ),
  .dpra( raddr ), .dpo( q1[6] )
);

prim_ram16x1d flag7(
  .wclk( wclk ), .we( we ), .a( waddr ),
  .d( d[7] ), .spo( ),
  .dpra( raddr ), .dpo( q1[7] )
);

reg [7:0] q;
always @(posedge rclk or posedge reset) begin
  if (reset) begin
    q <= 0;
  end
  else begin
    q <= q1;
  end
end


endmodule
```